Test 1: Recursive File Structure Report

The Problem:

1. Organise a recursive file structure into a relational database
2. Search database through web interface and return full path values of matching records

1. Organise a recursive file structure into a relational database

Representing hierarchical data in a relational database:

The 2 solutions considered were adjacency lists and nested sets.

Nested sets involve storing the first and second time each item in the hierarchical data is visited during a pre-order traversal. Each item is represented as a record with 2 fields to represent position. This implementation allows cheap searches for ancestors, but has expensive move, insert, and delete operations.

Adjacency lists also represent each item as a record, where 1 field is used to represent the parent node of the item. Only the root node will have a null parent node. This method has cheap move insert, and delete operations, but is expensive to find ancestry.

Within the context of this problem, nested sets would perform better assuming no additional metadata is stored about the hierarchical data, as searching ancestry will be needed for returning the full path values of each matching item. But if the full path is stored as an additional field, adjacency lists will perform the same as nested sets, as well allow for inexpensive modification.

The MYSQL tables created to store the hierarchical data:

```
CREATE TABLE Directories (DirectoryID int PRIMARY KEY NOT NULL, Name varchar(255), FullPath varchar(255), ParentID int references Directories(DirectoryID));

CREATE TABLE Files (FileID int PRIMARY KEY NOT NULL, Name varchar(255), FullPath varchar(255), ParentID int references Directories(DirectoryID));
```

Separate tables were created for files and directories as they may require different metadata to be stored.

Format for input data:

Input data is in the form of text files with new lines to denote a new item, and tab characters to represent the hierarchy. Special characters contained in this file must be escaped before being parsed. This implementation assumes that any leaf node in the hierarchical data is a file. The file is loaded into an array line by line unformatted.

Translating array to tree:

This part required an iterative algorithm and a basic tree data structure that is back linked to the parent nodes, and the previous node is stored on each iteration. The logic can be broken down into 3 parts:

- If previous item is less indented than current item, add the current node as a child of the previous item.
- If the previous item is more idented, set a new variable to the previous node, and call the parent node on that new variable as many times as the difference of indentation + 1. This will retrieve the parent node of the current item. The current item is then added as a child of the parent node.
- Otherwise, add the current item as a child of the previous node's parent. As the previous node and current node will be siblings.

Populating the database:

This part of the problem involved a depth first search of the created tree. Work was done pre-order as parent node needs to exist within the database before its children are added, as the primary key of the parent node needs to be known. This was achieved through having a global auto incremented value and the parsing the parentID on each new call of the depth first search. The path taken during the search is also stored and is used to enter the full path of each entry into the database.

2. Search the database to return full path of each matching item

This part of the problem involves a simple query as the full path of each item is stored in the database. Due to time constraints, I didn't have enough time to implement a web interface for queries, so instead I decided to make a simple console application that searches what the user inputs.

Instructions for running code:

- Add mysql connector/J to build path
- Set 4 constants in DatabaseCredentials.java to desired mysql database.
- Run LoadData.java to populate database with data from src/input.txt
- Run Search.java to perform queries to the database through console